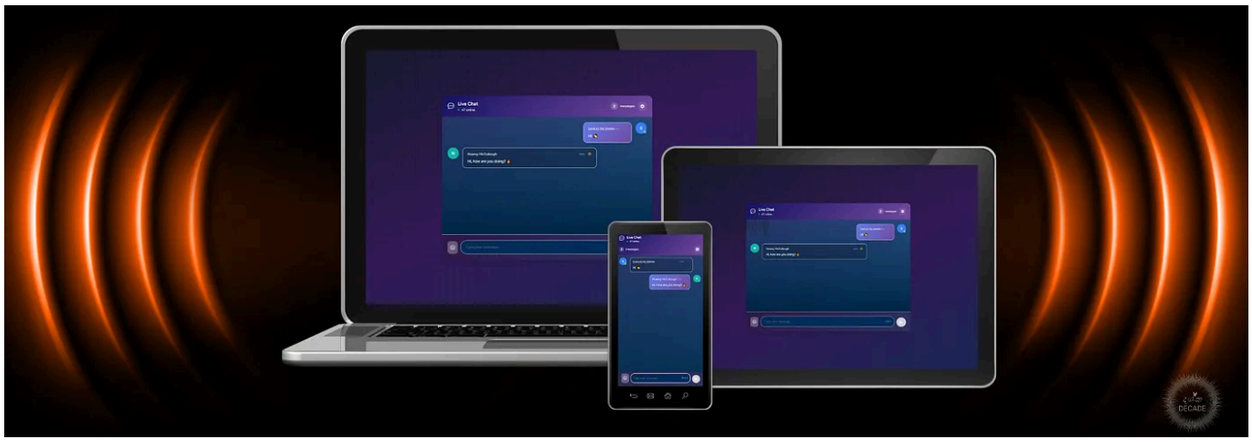
Building a Real-Time Chat Application with Laravel Reverb, Vue.js, and Inertia.

Sanju Dilshan Follow 7 min read · Aug 3, 2025



> *In this tutorial, we'll walk through the process of building a real-time chat application using **Laravel Reverb**, a first-party WebSocket server for Laravel, combined with **Laravel Breeze**, **Vue.js**, and **Inertia.js**. By the end, you'll have a fully functional chat app with real-time messaging, user authentication, and a clean UI.*

*This guide assumes you have basic knowledge of Laravel, Vue.js, and JavaScript, as well as **Node.js** and **Composer** installed on your system. Let's dive in!*

Prerequisites

Before we start, ensure you have the following installed:

> ***PHP** (>= 8.1)*
>
> ***Composer***

> *Node.js and npm*
>
> *MySQL or another database supported by Laravel*
>
> *A basic understanding of Laravel, Vue.js, and WebSocket concepts.*

Step 1: Set Up a New Laravel Project

First, create a new Laravel project named chat-app using Composer.

```
composer create-project laravel/laravel chat-app
```

Navigate to the project directory:

```
cd chat-app
```

Step 2: Install Laravel Breeze for Authentication.

We'll use Laravel Breeze to set up authentication with a Vue.js and Inertia.js frontend. Install Breeze:

```
composer require laravel/breeze --dev
```

Run the Breeze installation command and select Inertia with Vue when prompted:

```
php artisan breeze:install
```

> *Choose the following options during the setup:*
>
> ***Inertia.js** (Vue.js)*

> *Yes for TypeScript (optional)*
>
> *Yes for SSR (Server-Side Rendering, optional)*

Next, install the Node dependencies and build the frontend assets:

```
npm install
npm run dev
```

Step 3: Install and Configure Laravel Reverb.

Laravel Reverb is a WebSocket server for real-time communication. Install it via Composer

```
php artisan install:broadcasting
```

This command sets up the necessary configuration files and installs the required Node packages.

Next, configure the .env file with Reverb settings. Add or update the following:

```
BROADCAST_CONNECTION=reverb
REVERB_APP_ID=548513
REVERB_APP_KEY= GENERATED CODE
REVERB_APP_SECRET=GENERATED CODE
REVERB_HOST="localhost"
REVERB_PORT=8080
REVERB_SCHEME=http

VITE_REVERB_APP_KEY="${REVERB_APP_KEY}"
VITE_REVERB_HOST="${REVERB_HOST}"
VITE_REVERB_PORT="${REVERB_PORT}"
VITE_REVERB_SCHEME="${REVERB_SCHEME}"
```

```
# Replace with your computer's IPv4 address
VITE_APP_URL="http://192.168.8.100:8000"
VITE_HOST=192.168.8.100
```

*In your **bootstrap.js** update like that*

```
import axios from 'axios';
window.axios = axios;

window.axios.defaults.headers.common['X-Requested-With'] = 'XMLHttpRequest';

import Echo from 'laravel-echo';
import Pusher from 'pusher-js';


window.Pusher = Pusher;

window.Echo = new Echo({
    broadcaster: 'reverb',
    key: import.meta.env.VITE_REVERB_APP_KEY,
    wsHost: import.meta.env.VITE_REVERB_HOST,
    wsPort: import.meta.env.VITE_REVERB_PORT,
    wssPort: import.meta.env.VITE_REVERB_PORT,
    forceTLS: (import.meta.env.VITE_REVERB_SCHEME ?? 'https') === 'https',
    disableStats: true,
});
```

*Update the Vite configuration in **vite.config.js** to ensure the frontend connects to the correct server:*

```
import { defineConfig } from 'vite';
import laravel from 'laravel-vite-plugin';
import vue from '@vitejs/plugin-vue';

export default defineConfig({
    plugins: [
        laravel({
            input: 'resources/js/app.js',
            ssr: 'resources/js/ssr.js',
            refresh: true,
        }),
        vue({
            template: {
                transformAssetUrls: {
                    base: null,
                    includeAbsolute: false,
                },
            },
        }),
```

```
    ],
    server: {
        host: '192.168.8.100',
        cors: {
            origin: 'http://192.168.8.100:8000',
            methods: 'GET,HEAD,PUT,PATCH,POST,DELETE',
            preflightContinue: false,
            optionsSuccessStatus: 204,
        },
    },
});
```

Run the Reverb server in a separate terminal:

```
php artisan reverb:start
```

Step 4: Create the Message Model and Migration

Create a Message model with a migration:

```
php artisan make:model Message -m
```

Update the migration file(database/migrations/xxxx_create_messages_table.php) to include user_id and message fields:

```
Schema::create('messages', function (Blueprint $table) {
    $table->id();
    $table->foreignId('user_id')->constrained()->onDelete('cascade');
    $table->text('message');
    $table->timestamps();
});
```

```
php artisan migrate
```

Define the Message model (app/Models/Message.php):

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Message extends Model
{
    use HasFactory;

    protected $fillable = ['user_id', 'message'];

    public function user()
    {
        return $this->belongsTo(User::class);
    }
}
```

Update the User model (app/Models/User.php) to include a relationship with messages:

```php
public function messages()
{
    return $this->hasMany(Message::class);
}
```

Step 5: Create the MessageSent Event

Create a broadcast event for sending messages:

```
php artisan make:event MessageSent
```

Update the MessageSent event (app/Events/MessageSent.php):

```php
<?php

namespace App\Events;

use App\Models\Message;
use App\Models\User;
use Illuminate\Broadcasting\Channel;
use Illuminate\Broadcasting\InteractsWithSockets;
```

```php
use Illuminate\Contracts\Broadcasting\ShouldBroadcastNow;
use Illuminate\Foundation\Events\Dispatchable;
use Illuminate\Queue\SerializesModels;
use Illuminate\Support\Facades\Log;

class MessageSent implements ShouldBroadcastNow
{
    use Dispatchable, InteractsWithSockets, SerializesModels;

    public $user;
    public $message;

    public function __construct(User $user, Message $message)
    {
        $this->user = $user;
        $this->message = $message;
    }

    public function broadcastOn(): array
    {
        Log::info('Broadcasting message sent event', [
            'user_id' => $this->user->id,
            'message_id' => $this->message->id,
        ]);

        return [new Channel('chat')];
    }

    public function broadcastAs(): string
    {
        return 'message.sent';
    }

    public function broadcastWith(): array
    {
        return [
            'user' => [
                'id' => $this->user->id,
                'name' => $this->user->name,
            ],
            'message' => [
                'id' => $this->message->id,
                'message' => $this->message->message,
                'created_at' => $this->message->created_at->toISOString(),
            ],
        ];
    }
}
```

This event broadcasts messages to the chat channel in real-time using Reverb.

Step 6: Create the Chat Controller

Generate a controller for handling chat-related requests:

```
php artisan make:controller ChatController
```

Update the ChatController (app/Http/Controllers/ChatController.php):

```
<?php

namespace App\Http\Controllers;

use App\Events\MessageSent;
use App\Models\Message;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;
use Inertia\Inertia;
use Illuminate\Support\Facades\Log;

class ChatController extends Controller
{
    public function index()
    {
        return Inertia::render('ChatApp', [
            'userId' => Auth::id(),
        ]);
    }

    public function getMessages()
    {
        $messages = Message::with('user:id,name')
            ->orderBy('created_at', 'asc')
            ->take(100)
            ->get();

        return response()->json($messages);
    }

    public function sendMessage(Request $request)
    {
        $request->validate([
            'message' => 'required|string|max:1000',
        ]);

        $user = Auth::user();

        if (!$user) {
            return response()->json(['error' => 'Unauthorized'], 401);
        }

        $message = $user->messages()->create([
            'message' => $request->message,
        ]);

        $message->load('user:id,name');

        Log::info('Message sent start');
        MessageSent::dispatch($user, $message);
        Log::info('Message sent ok');
```

```
        return response()->json([
            'status' => 'Message Sent!',
            'message' => $message,
        ]);
    }
}
```

Step 7: Define Routes.

Update routes/web.php to include routes for the chat application:

```
<?php

use App\Http\Controllers\ChatController;
use Illuminate\Support\Facades\Route;
use Illuminate\Support\Facades\Auth;

Route::middleware('auth')->group(function () {
    Route::get('/chat', [ChatController::class, 'index'])->name('chat');
    Route::get('/messages', [ChatController::class, 'getMessages']);
    Route::post('/messages', [ChatController::class, 'sendMessage']);
});
```

Step 8: Create the Chat Frontend with Vue.js

Create a Vue component for the chat interface at resources/js/Pages/ChatApp.vue:

```
<template>
  <div class="chat-wrapper">
    <div class="messages-window" ref="messagesContainer">
      <div v-if="messages.length === 0" class="no-messages">
        <p>No messages yet</p>
      </div>

      <div v-for="msg in messages" :key="msg.id" class="message" :class="{ 'own'
        <div class="message-content">
          <strong>{{ msg.user?.name || 'Unknown' }}</strong>
          <div class="text" v-html="formatMessage(msg.message)"></div>
        </div>
      </div>
    </div>

    <div class="input-area">
      <input v-model="newMessage" @keyup.enter="sendMessage" placeholder="Type y
      <button @click="sendMessage" :disabled="!newMessage.trim()">Send</button>
    </div>
  </div>
</template>
```

```
const props = defineProps({
  userId: Number
});

const currentUserId = ref(props.userId);
const messages = ref([]);
const newMessage = ref('');
const messagesContainer = ref(null);

const loadMessages = async () => {
  try {
    const res = await axios.get('/messages');
    messages.value = res.data;
    scrollToBottom();
  } catch (err) {
    console.error('Failed to load messages:', err);
  }
};

const sendMessage = async () => {
  if (!newMessage.value.trim()) return;

  try {
    const res = await axios.post('/messages', {
      message: newMessage.value.trim()
    });
    newMessage.value = '';
  } catch (err) {
    console.error('Failed to send message:', err);
  }
};

const scrollToBottom = () => {
  nextTick(() => {
    if (messagesContainer.value) {
      messagesContainer.value.scrollTop = messagesContainer.value.scrollHeight;
    }
  });
};

const formatMessage = (message) => {
  return message.replace(/(https?:\/\/[^\s]+)/g, '<a href="$1" target="_blank">$
};

onMounted(() => {
  loadMessages();
  if (window.Echo) {
    window.Echo.channel('chat')
      .listen('.message.sent', (e) => {
        messages.value.push({
          id: e.message.id,
          user: e.user,
          message: e.message.message,
          created_at: e.message.created_at
        });
        scrollToBottom();
      });
  }
});

onUpdated(() => {
  scrollToBottom();
```

```
});
</script>

<style scoped>
.chat-wrapper {
  max-width: 600px;
  margin: 40px auto;
  display: flex;
  flex-direction: column;
  border: 1px solid #ddd;
  border-radius: 8px;
  overflow: hidden;
  font-family: sans-serif;
  background: #fff;
}

.messages-window {
  flex: 1;
  height: 400px;
  overflow-y: auto;
  padding: 16px;
  background: #f9fafb;
}

.message {
  margin-bottom: 12px;
  max-width: 80%;
}

.message.own {
  margin-left: auto;
  text-align: right;
}

.message-content {
  background: #e5e7eb;
  padding: 10px 14px;
  border-radius: 8px;
}

.message.own .message-content {
  background: #dbeafe;
}

.text {
  margin-top: 4px;
}

.input-area {
  display: flex;
  border-top: 1px solid #eee;
}

.input-area input {
  flex: 1;
  padding: 10px;
  border: none;
  outline: none;
}

.input-area button {
  padding: 10px 16px;
  border: none;
  background: #3b82f6;
```

```
    color: white;
    cursor: pointer;
  }

  .no-messages {
    text-align: center;
    padding: 40px;
    color: #6b7280;
  }
  </style>
```

Step 8:Test the Application

Start the Laravel development server with your local ipv4 address:

```
php artisan serve --host=192.168.8.104 --port=8000
```

Ensure the Reverb server is running:.

```
php artisan reverb:start --port=8080
```

Run Vite to compile frontend assets:

```
npm run dev
```

Test Across Two Devices on the Same Network

To experience the real-time chat, test the application using two devices (e.g., a computer and a smartphone, or two computers) connected to the same Wi-Fi or local network.

1. **Connect Devices to the Same Network**:

- Ensure both devices are on the same Wi-Fi network as the computer hosting the Laravel app (IP: 192.168.8.104).

- Verify your computer's IPv4 address by running ipconfig (Windows) or ifconfig/ip addr (macOS/Linux) if 192.168.8.104 doesn't work.

1. **Open Browsers on Each Device**:

- On the first device (e.g., your computer), open a browser (e.g., Chrome) and navigate to http://192.168.8.104:8000.
- On the second device (e.g., a smartphone or another computer), open a browser and go to the same URL: http://192.168.8.104:8000.

You should see a clean chat UI where authenticated users can send and receive messages in real-time. Messages are persisted in the database and broadcast via Reverb to all connected clients.

Troubleshooting Tips

- **Reverb Connection Issues**: Ensure the REVERB_* environment variables match your setup, and the Reverb server is running.
- **CORS Errors**: Verify the vite.config.js CORS settings match your app's URL.
- **Messages Not Loading**: Check the browser console for errors and ensure the Echo client is properly initialized in resources/js/bootstrap.js.

Conclusion

You've successfully built a real-time chat application using **Laravel Reverb**, **Vue.js**, and **Inertia.js**! This app demonstrates the power of Laravel's first-party WebSocket solution for real-time communication. You can extend it by adding features like private messaging, typing indicators, or message deletion.

Feel free to share your thoughts or questions in the comments below, and happy coding!

Thanks for reading ....!

Written by Sanju Dilshan

7 followers · 10 following

Follow

Web development enthusiast & student at the University of Jaffna. Passionate about building modern web apps & sharing my coding journey.

No responses yet

Write a response

What are your thoughts?

More from Sanju Dilshan

Sanju Dilshan

Supercharge Laravel Debugging with Telescope

If you’re looking to monitor, debug, and dive deep into your Laravel application’s...

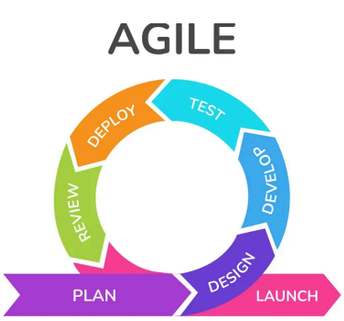
Apr 20, 2025

Sanju Dilshan

How To install Tailwind CSS in a React with Vite project

How To install & Setup Tailwind CSS in a React with Vite project

Feb 22, 2025 1



Sanju Dilshan

The evolution of Database Management system

What is DBMS

Jul 20, 2022

Sanju Dilshan

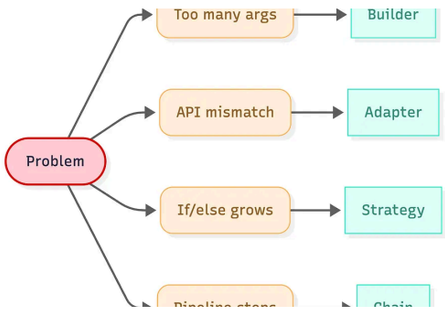
Agile Methodology

Agile is a project management and product development approach that emphasizes...

Aug 13, 2023 1

See all from Sanju Dilshan

Recommended from Medium

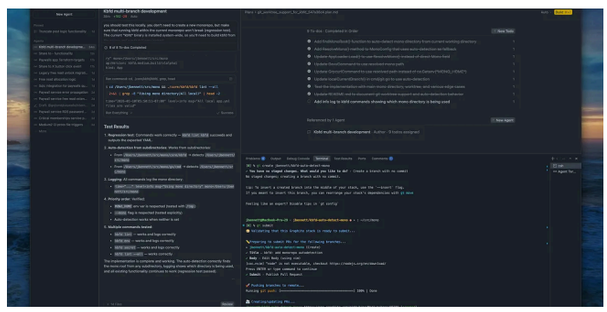


In Women in Technology by Alina Kovtun ✨

Stop Memorizing Design Patterns: Use This Decision Tree Instead

Choose design patterns based on pain points: apply the right pattern with minimal over-...

Jan 29 3.8K 32



Jacob Bennett

The 5 paid subscriptions I actually use in 2026 as a Staff Software...

Tools I use that are (usually) cheaper than Netflix

Jan 18 3K 77

Shivansh Talwar

In Towards AI by Adham Khaled

Implementing Secure Session-Based Authentication in Nuxt 4...

A step-by-step guide to building secure session-based authentication in Nuxt 4 usin...

Aug 19, 2025 1

I Cancelled My ~$200/mo Claude API Subscription, Again.

Kimi K2.5 didn't just lower the price. It destroyed the business model of "renting...

Feb 8 1.6K 48

In Startup Stash by Zack Liu

7 "Boring" Micro-SaaS Ideas Making $2k/Month (The...

No AI hype. No complex algos. Just solving rich people's problems.

Jan 2 1.2K 50

Developer Awam

What is Hugging Face? And Why It Matters for Laravel Developers

Discover Hugging Face and why it's a game-changer for Laravel developers to build...

Sep 16, 2025 47

See more recommendations